Map-Reduce 课程设计 之

《日志统计分析》实验报告

组号 2018st21

小组成员 151220014 陈越 & 151220017 陈紫琦

1 实验要求

本课程设计通过使用 MapReduce 来实现日志分析。请按照以上给定的日志文件格式，按照以下的要求进行分析统计 ：

任务 1：统计日志中各个状态码（200，404，500）出现总的频次，并且按照小时时间窗，输出各个时间段各状态码的统计情况。

任务 2：统计每个 IP 访问总的频次，并且按照小时时间窗，输出各个时间段各个 IP 访问的情况。

任务 3：统计每个接口(请求的 URL)访问总的频次，并且以接口为文件，按照秒为单位的时间窗，输出各个时间段各接口的访问情况。

任务 4：统计每个接口的平均响应时间，并且以接口为分组，按照小时时间窗，输出各个时间段各个接口平均的响应时间。

任务 5：接口访问频次预测，给 2015-09-08.log 到 2015-09-21.log 共 14 天的日志文件，作为训练数据，设计预测算法来预测下一天（2015-09-22）每个小时窗内每个接口（请求的 URL）的访问总频次。将该结果与当天实际的统计值(2015-09-22.log)做 RMSE 验证。

2 程序设计的主要流程

2.1 任务 1-4

图 1 任务 1-4 主要流程

2.2 任务 5

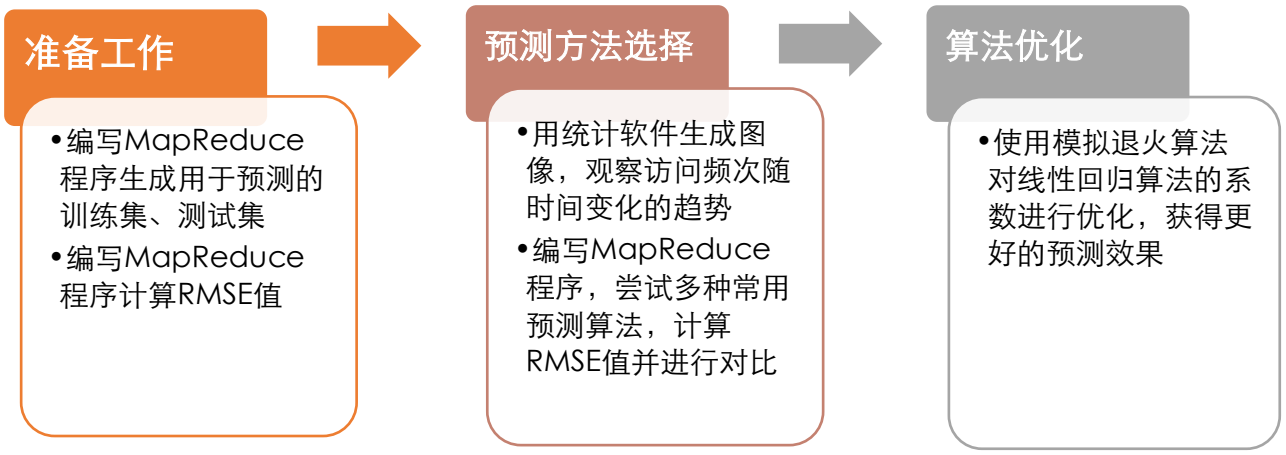

图 2 任务 5 主要流程

3 程序采用的主要算法

3.1 任务 1-4

任务 1-4 的实现代码位于 src/main/java/LogAnalysis.java 中，经过对原始数据文件 2015-09-08.log 的一遍扫描，生成任务 1-4 所需的输出文件。具体算法及代码细节介绍如下。

3.1.1 数据输入输出格式

数据输入格式采用默认的 TextInputFormat 类。由于实验要求多个文件夹输出且文件名以".txt"结尾且不能有如"-r-00000"的字符串出现，因此采用 MultipleOutputs 类将数据输出到多个文件夹和文件，并自定义类 LogTextOutputFormat（继承自 TextOutputFormat），重写其中的 getDefaultWorkFile 方法从而更改输出文件的命名方式，代码如下：

```
public static class LogTextOutputFormat extends TextOutputFormat<Text, Text>
{
    @Override
    public Path getDefaultWorkFile(TaskAttemptContext context, String extension) {
        return new Path(getOutputName(context)+".txt");
    }
}
```

另外，由于输出文件要求 key 和 value 值之间以空格分隔，而默认以制表符"\t"分隔，还需要将 configuration 中默认的分隔符修改，代码如下：

```
conf.set("mapred.textoutputformat.ignoreseparator", "true");
conf.set("mapred.textoutputformat.separator", " ");
```

3.1.2 Mapper

根据题目要求，从输入的每行数据中提取出状态码、时间、IP、接口、响应时间等信息，由于每个任务需要输出的信息不同，为了标识每个任务发射的信息，在发射的 key 上增加数字前缀（1、2、3、4）表示对应的任务序号。设置 Mapper 输出的 key 和 value 值类型都为 Text 类，各任务发射的键值对信息如下列伪代码所示：

```
Emit <"1_00:00-00:00_stateNo", "1">
Emit <"1_time_stateNo>, "1">
Emit <"2_ip", "1">
Emit <"2_ip_time", "1">
Emit <"3_interface", "1">
Emit <"3_interface_time", "1">
Emit <"4_interface", "responseTime_1">
Emit <"4_interface_time">, "responceTime_1">
```

其中，任务 1 通过特殊的时间标记 "00:00-00:00"以在 reduce 时统计每个状态码对应的总频次并保证一定在输出文件的开头显示。任务 2、3 较为类似，都发送只带 IP 或接口的信息以统计访问总频次，以及加上时间窗的信息以统计每个时间窗内的访问总频次，并且两种发送方式 key

的前缀相同保证了后续在 Reducer 上排序时总频次一定在文件的开头。但需注意的是任务 2 以小时为时间窗，而任务 3 以秒为时间窗。任务 4 与前两个任务的区别是需要统计响应时间的平均值，模仿 k-means 算法中计算均值的过程，在 value 中将响应时间和计数 1 组合并发射。

3.1.3 Combiner

作为本地的 Reducer，Combiner 需要先通过接收到的 key 值的前缀判断对应的任务序号，对于任务 1、2、3，只需仿照 WordCount 程序将相同 key 对应的 value 值求和即可。对于任务 4，则需仿照 k-means 算法计算相同 key 值对应的响应时间的均值和个数并发射。

3.1.4 Partitioner

将 key 值以"_"分割，如果分割后个数为 2，则直接按 key 值的默认的哈希值划分；若分隔后个数为 3，则按 key 值中的前两个信息的 hash 值划分，以确保相同输出文件中的键值对被划分到同一个 reduce 节点。

3.1.5 Reducer

先通过接收到的 key 值的前缀判断对应的任务序号。对于任务 1，设置私有成员变量：

```
private Integer count200=0;//统计当前 item 下 200 出现的次数
private Integer count404=0; //统计当前 item 下 200 出现的次数
private Integer count500=0; //统计当前 item 下 200 出现的次数
static String CurrentItem = new String();//标记当前 item<1_hourInfo>
```

然后在 reduce 方法中，仿照倒排索引程序统计每个时间窗对应的状态码出现频次，伪代码如下：

```
若新的 key 的"1_hourInfo"与 currentItem 相同
  则统计对应状态码的值，并赋值给 count200/count404/count500;
否则
  若 CurrentItem=="1_00:00-00:00"//表示状态码出现的总频次
      输出<200: , count200>,<404: , count404>,<500: , count500>
  否则
      输出<hourInfo: , count200+" "+count404+" "+count505>
```

并在 cleanup 方法中将最后一个 key 值对应的输出。

对于任务 2、3、4，统计方法与 Combiner 中类似，但需要注意输出到特定的文件夹中，文件夹名从 key 值中的第二个信息可获得。

3.2 任务 5

3.2.1 生成训练数据集和测试数据集

由于任务 5 需要统计的数据为每个小时窗内每个接口的访问频次信息，仿照任务 3 的代码，将以秒为单位的时间窗改为以小时为单位的时间窗。并且因为某些时间窗内接口访问频次可能为 0，定义数据类型 TimeTable 并实现 Writable 接口，以保存某个接口对应的访问频次表，该表中记录了 9 月 8 号至 21 号每个时间段对应的访问频次信息，并初始化为全 0。设置 Mapper 的 value 输出类型以及 Reducer 的 value 输入类型都为 TimeTable 类，以接口名作为 key 值，发射访问频次表，并在 combiner 和 reducer 阶段合并该表格，最终输出为每个接口对应一个文件，其格式如下（以 tour-category-ids-query.txt 为例）：

表格 1 训练数据格式示例

时间窗	9 月 8 日	9 月 9 日	9 月 10 日	9 月 11 日	9 月 12 日	……
00:00-01:00	9684	9774	10454	9755	10514	……
01:00-02:00	15039	14366	13003	13456	14205	……
02:00-03:00	9563	10311	8796	8334	9741	……
03:00-04:00	7507	7486	7048	6476	6586	……
04:00-05:00	8745	8698	7456	7440	7449	……
05:00-06:00	8806	7903	6371	6242	6433	……
06:00-07:00	8334	7276	7710	7565	8538	……
07:00-08:00	9238	9174	9732	8707	10384	……
08:00-09:00	22101	20891	20926	19906	16103	……
……	……	……	……	……	……	……

另外，还需注意输入文件夹中包括 9 月 8 日-22 日所有日志文件，训练数据需要将 22 号的日志数据排除在外，采用 FileSystem 类的 globStatus 方法以及正则表达式将 9 月 22 日的文件过滤掉，代码如下：

```
FileStatus[] fileStatusArray = fileSystem.globStatus(new Path(args[0]+"/*.log"), new
PathFilter() {
    public boolean accept(Path path) {
        String regex = "^(?!.*?2015-09-22).*$";
        return  path.toString().matches(regex);
    }
});
```

测试数据集的生成过程与训练数据集类似，但只输入 9 月 22 日的日志文件。

3.2.2 预测算法

经过多种算法的尝试，我们最终采用了线性回归算法，对每个时间段，赋予系数，其中的计算及优化方法将在下一节中介绍，并利用二维数组 double Co[24][15]存放不同时间的不同系数。预测时，在 Mapper 中对于每个接口的每个时间段的预测值 y，计算得：

并直接输出到文件中，格式与测试数据集相同，供后续计算 RMSE 值时使用。

3.2.3 计算 RMSE 值

将预测后的输出的结果文件夹和测试数据集所在的文件夹作为输入。

在 Map 阶段，从文件名中获得接口信息（url），从预测结果或测试数据中的每行数据中获得信息时间窗(time)和访问频次（num），发射键值对<time_url,num>。

Partition 阶段，按照 time 进行划分，确保同一段时间被划分到同一个 reduce 节点。

Reduce 阶段，定义成员变量如下：

```
private double rmse=0;//用于最终计算 rmse
private double currentDis=0;//用于统计当前时间的均方差
private double urlcount=0;//用于统计总的 url 数
private String currentTime=new String();//用于记录当前的时间
```

对于相同的 key,计算 dis = (value[1]-value[0])^2，若新的时间 time 与 currentTime 不同，则按下式更新：

```
rmse+=Math.sqrt(currentDis/urlcount);
Currentdis=dis;urlcount=1;
```

否则：

```
Currentdis+=dis;urlcount++;
currentTime=time;
```

最后，在 cleanup 方法中，更新最后一个时间段的 rmse 值，并输出最终结果到文件中：

```
rmse+=Math.sqrt(currentDis/urlcount);
rmse=rmse/24;
context.write(new Text("rmse:"),new Text(Double.toString(rmse/24)));
```

4 进行的优化工作及效果

4.1 预测算法选择

在生成训练数据集后，我们利用 Excel 绘制出了对各接口的每个时间段的访问频次随日期变化的趋势图、每日访问频次随时间窗变化的图像以及连续 14 天访问频次随时间变化的总趋势图，以接口 tour-category-ids-query 为例：

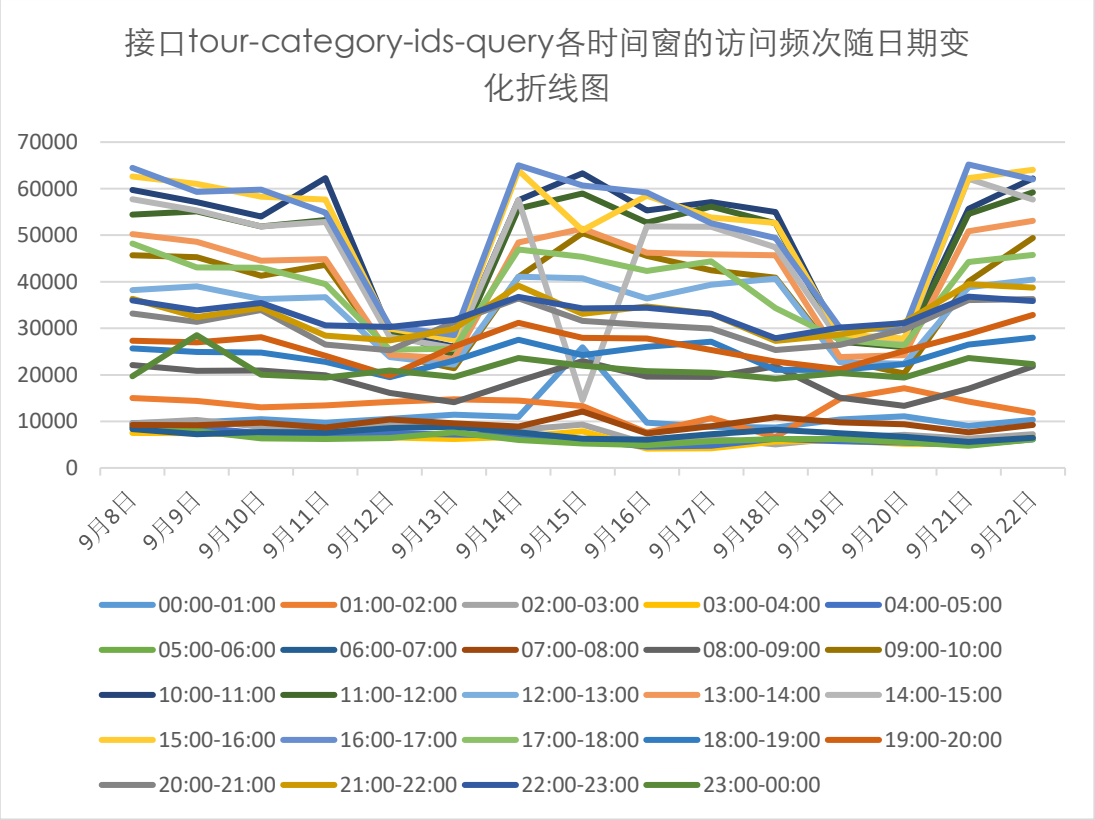

图 3

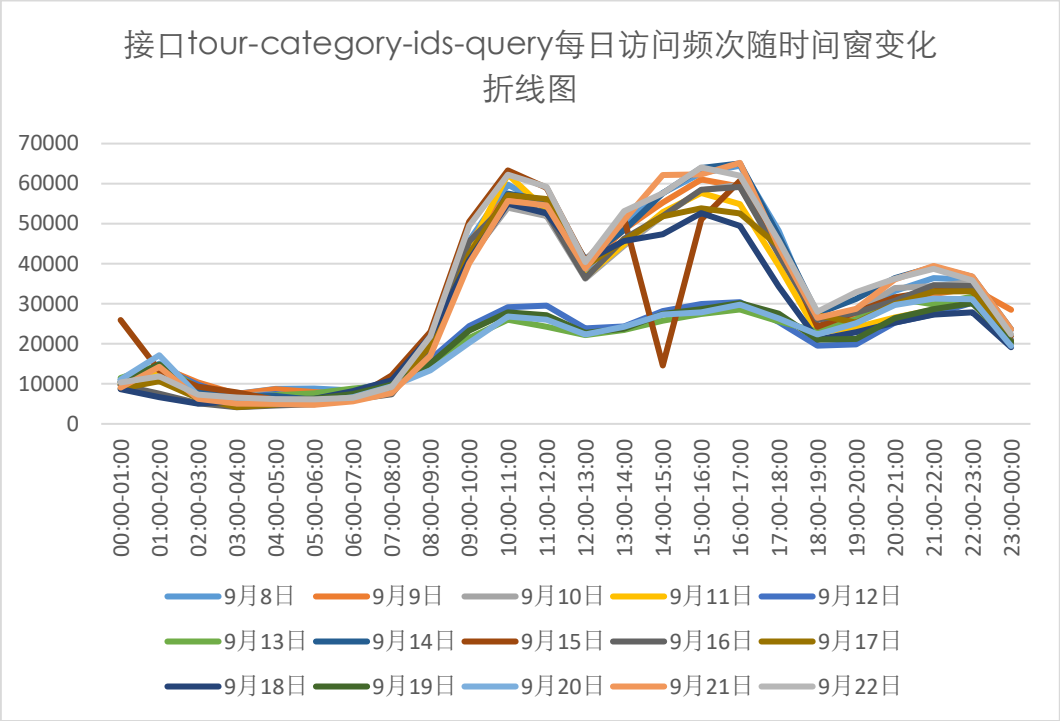

图 4

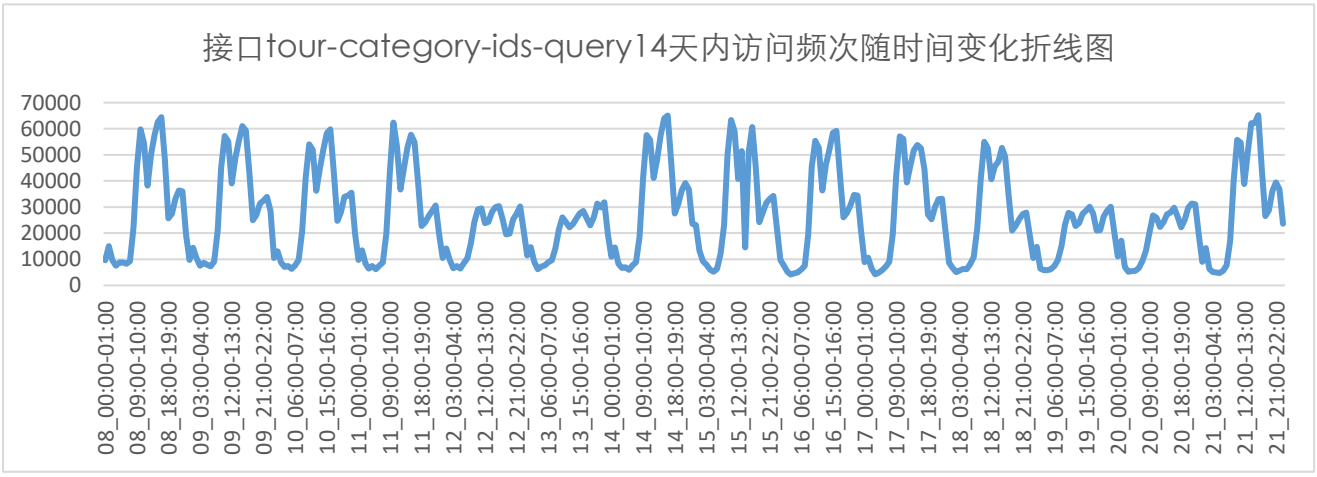

图 5

由以上三张图可见，对于同一个接口，其访问频次基本以 24 小时为周期进行周期性变化，也就是说对于每天的同一个时间段，其访问频次较为相似。但不能排除某些天访问频次的峰值水平处于比较低的位置。根据该特征以及查阅的相关资料，我们最先尝试的是平均法预测，包括简单平均法、移动平均法、加权平均法、指数平滑法。

由于某些接口的时间段访问频次随日期变化的曲线有类似于直线或曲线变化的趋势，我们又尝试了一般的回归方法如线性回归、多项式回归（最小二乘法），以及预测时间序列常用的模型

ARIMA，但直接使用的效果都一般。最后我们采用了使用模拟退火算法优化线性回归系数的方法，取得了较好预测效果。

4.2 预测算法介绍

本次实验尝试过的预测算法位于包 Predictor 中，并均实现了接口 Predictor，在类 LogPredict 的 main 方法中任意更改预测算法或的预测结果。各预测算法介绍如下：

a) 简单平均法

对于每个接口的每个时间段，直接将前 14 天历史访问频次的平均值作为 9 月 22 日的预测值。

b) 移动平均法

在简单平均的基础上，只取最近的 n 天（1≤n≤14）的平均值作为预测值。

c) 加权平均法

在简单平均的基础上，为每天的访问频次数据赋予权重 w1,.....,w14，其中 w1+.....+w14=1。一般设为越近的日期权重越大。

d) 指数平滑法

指数平滑法一种特殊的加权移动平均法。它加强了观察期近期观察值对预测值的作用，对不同时间的观察值所赋予的权重不等，从而加大了近期观察值的权重，权重之间按等比级数减少，此级数的首项为平滑常数 α，公比为(1- α)。并且，指数平滑法对于观察值所赋予的权数有伸缩性，可以取不同的 α 值以改变权数的变化速率。

其预测公式为：

其中为 t+1 时刻的预测值，为 t 时刻的预测值，为 t 时刻的实际值，α为平滑常数。具体实现代码如下：

```
double result = Integer.parseInt(line[1]);
for(int i = 2; i <= 14; i++)
        result= alpha*Integer.parseInt(line[i])+(1-alpha)*result;
```

e) 最小二乘法

最小二乘法是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配假设给定的数据点和其对应的函数值为 (x1, y1), (x2, y2), ... (xm, ym)，需要做的就是得到一个多项式函数 f(x) = a0 * x + a1 * pow(x, 2) + .. + an * pow(x, n)，使其对所有给定 x 所计算出的 f(x)与实际对应的 y 值的差的平方和最小，也就是计算多项式的各项系数 a0, a1, ... an。apache-commons-math3 是 java 的一种科学计算类库，使用其中的 PolynomialCurveFitter 类，设置好拟合的多项式阶数和权重后可直接得到预测模型。

f) 优化的线性回归法

在预处理阶段，以指数平滑的预测值作为 y 值，首先通过梯度下降法求解一个近似的系数 Co。然后将所有数据除以 10000,去除比较小的数据的影响。梯段下降算法实现公式以及伪代码如下所示：

$$\theta_j=\theta_j+\alpha\sum_{i=1}^{m}(y^{(i)}-h_\theta(x^{(i)}))x_j^{(i)}$$

```
gradientDescentLoop
        for i=0 to m
           double [] partial_derivative = compute_partial_derivative();//求解每个 i 的偏导数
           theta[i]-= alpha * partial_derivative[i];
compute_partial_derivative()
    double [] partial_derivative = new double[theta.length];
    for(int j =0;j<theta.length;j++)//遍历，对每个 theta 求偏导数
          partial_derivative[j]= compute_partial_derivative_for_theta(j);//对 theta j 求偏导
    return partial_derivative;
compute_partial_derivative_for_theta(int j)
    double sum=0.0;
    for(int i=0;i<row;i++)//遍历 每一行数据
          sum+=h_theta_x_i_minus_y_i_times_x_j_i(i,j);
    return sum/row;
h_theta_x_i_minus_y_i_times_x_j_i(int i,int j)
     double[] oneRow = getRow(i);//取一行数据，前面是 feature，最后一个是 y
     double result = 0.0;
     for(int k=0;k< (oneRow.length-1);k++)
```

```
            result+=theta[k]*oneRow[k];//计算线性回归的预测值
        result-=oneRow[oneRow.length-1];//真实值
        result*=oneRow[j];
        return result;
```

下一步通过模拟退火法，更新优化系数 Co。具体实现上，对 24 个时间段分别执行模拟退火的循环，设置初值如下：

```
eps =0.1;        //搜索停止条件阀值
delta= 0.98;     //温度下降速度
T = 1;           //初始温度;
double rmse =3674.626;//初始 Co 所求的 rmse
private static double foot = 0.01;
```

模拟退火算法代码如下：

```
SimAnnLoop(time)
    while (T>eps)
        double[][] newCo=copyto(Co);
        for (int i=0;i<loopend;i++)//从当前 Co 的状态中随机生成 loopend 个邻域中的状态
        {
            for (int j=0;j<=14;j++)
            {//每个系数以 1/3 的概率保持不变，或者加上 foot，或者减去 foot
                int rand = random.nextInt();
                if (rand%3==1)
                    newWeight[time][j]=weight[time][j]+foot;
                else if (rand%3==2)
                    newWeight[time][j]=weight[time][j]-foot;
            }
            如何新的状态的 rmse 的值比当前状态好，则 Co=newCo;rmse=newrmse;
        否则以 Math.exp((newrmse-rmse)/T)的概率，更新 Co=newCo; ;rmse=newrmse;
            }
            T=T*delta;
        }
```

4.3 各预测算法实验效果对比

表格 2 各预测算法实验效果对比

预测算法	参数	RMSE 值
简单平均法	-	10861.82

移动平均法	n = 7	4600.69
	n = 3	5327.32
	n = 2	4303.18
	n = 1	3496.89
加权平均法	w2=2*w1,w3=3*w1,...,w14=14*w1(w1+...+w14=1)	6943.97
指数平滑法	α = 0.95	3534.03
	α = 0.87	3513.32
	α = 0.8	3528.18
最小二乘法	degree=1	9580.48
	degree=2	5404.72
	degree=3	8928.195
优化后的线性回归法	经模拟退火优化过的权值矩阵 Co[24][15]	688.69

5 程序的性能分析

表格 3 各任务运行时间和内存使用情况

任务	运行总时间(s)	物理内存使用情况(MB)	虚拟内存使用情况(MB)	堆使用情况(MB)
1-4	55	10597	44140	12077
Task5 生成训练集	139	109194	416880	132022
Task5 预测	19	16567	248701	50966

6 程序运行截图

6.1 任务 1-4

6.1.1 运行过程

```
18/07/20 11:51:55 INFO mapreduce.Job: Running job: job_1528693522781_6861
18/07/20 11:52:01 INFO mapreduce.Job: Job job_1528693522781_6861 running in uber mode : false
18/07/20 11:52:01 INFO mapreduce.Job:  map 0% reduce 0%
18/07/20 11:52:12 INFO mapreduce.Job:  map 20% reduce 0%
18/07/20 11:52:15 INFO mapreduce.Job:  map 32% reduce 0%
18/07/20 11:52:19 INFO mapreduce.Job:  map 36% reduce 0%
18/07/20 11:52:22 INFO mapreduce.Job:  map 45% reduce 0%
18/07/20 11:52:25 INFO mapreduce.Job:  map 50% reduce 0%
18/07/20 11:52:27 INFO mapreduce.Job:  map 50% reduce 4%
18/07/20 11:52:28 INFO mapreduce.Job:  map 56% reduce 4%
18/07/20 11:52:31 INFO mapreduce.Job:  map 64% reduce 4%
18/07/20 11:52:34 INFO mapreduce.Job:  map 73% reduce 4%
18/07/20 11:52:36 INFO mapreduce.Job:  map 77% reduce 8%
18/07/20 11:52:37 INFO mapreduce.Job:  map 92% reduce 8%
18/07/20 11:52:38 INFO mapreduce.Job:  map 96% reduce 8%
18/07/20 11:52:39 INFO mapreduce.Job:  map 100% reduce 29%
18/07/20 11:52:42 INFO mapreduce.Job:  map 100% reduce 65%
18/07/20 11:52:45 INFO mapreduce.Job:  map 100% reduce 77%
18/07/20 11:52:48 INFO mapreduce.Job:  map 100% reduce 100%
18/07/20 11:52:52 INFO mapreduce.Job: Job job_1528693522781_6861 completed successfully
```

图 6 任务 1-4 运行过程截图

6.1.2 作业运行状态

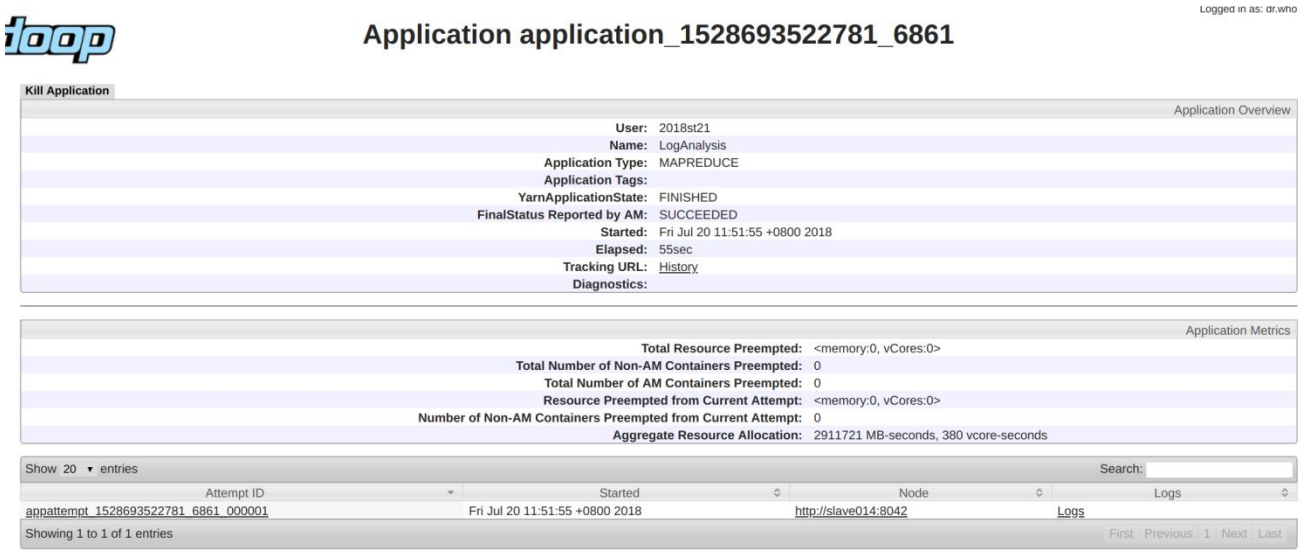

图 7 任务 1-4 作业运行状态截图

6.1.3 查看输出文件

```
200: 10403833
404: 4
500: 0
00:00-01:00  200:251501 404:0 500:0
01:00-02:00  200:417175 404:0 500:0
02:00-03:00  200:540917 404:0 500:0
03:00-04:00  200:700389 404:0 500:0
04:00-05:00  200:724747 404:0 500:0
05:00-06:00  200:377207 404:0 500:0
06:00-07:00  200:297346 404:0 500:0
07:00-08:00  200:309841 404:0 500:0
08:00-09:00  200:366154 404:0 500:0
09:00-10:00  200:403606 404:0 500:0
10:00-11:00  200:505789 404:2 500:0
11:00-12:00  200:449247 404:0 500:0
12:00-13:00  200:384767 404:0 500:0
13:00-14:00  200:439435 404:2 500:0
14:00-15:00  200:469977 404:0 500:0
15:00-16:00  200:482252 404:0 500:0
16:00-17:00  200:524866 404:0 500:0
17:00-18:00  200:500323 404:0 500:0
18:00-19:00  200:336887 404:0 500:0
19:00-20:00  200:336751 404:0 500:0
20:00-21:00  200:396558 404:0 500:0
21:00-22:00  200:392855 404:0 500:0
22:00-23:00  200:438798 404:0 500:0
23:00-00:00  200:356445 404:0 500:0
```

```
10.10.0.101: 50
00:00-01:00 4
02:00-03:00 10
03:00-04:00 1
04:00-05:00 3
05:00-06:00 2
13:00-14:00 7
17:00-18:00 7
18:00-19:00 13
22:00-23:00 3
```

图 8 任务 1、2 输出文件截图

```
tour-category-ids-query: 764394
00:00:00 2
00:00:01 9
00:00:02 7
00:00:03 6
00:00:04 2
00:00:05 7
00:00:06 8
00:00:07 7
00:00:08 7
00:00:09 5
00:00:10 2
00:00:11 1
00:00:12 3
00:00:13 8
00:00:14 3
00:00:15 8
00:00:16 6
00:00:17 3
```

```
tour-category-ids-query: 7.83
00:00-01:00 3.57
01:00-02:00 3.02
02:00-03:00 3.91
03:00-04:00 4.82
04:00-05:00 5.56
05:00-06:00 4.49
06:00-07:00 3.92
07:00-08:00 3.84
08:00-09:00 4.07
09:00-10:00 7.15
10:00-11:00 14.13
11:00-12:00 11.98
12:00-13:00 5.02
13:00-14:00 8.04
14:00-15:00 9.97
15:00-16:00 9.23
16:00-17:00 12.36
17:00-18:00 7.53
18:00-19:00 4.86
19:00-20:00 4.78
20:00-21:00 4.48
21:00-22:00 4.40
22:00-23:00 4.46
23:00-00:00 5.16
```

图 9 任务 3、4 输出文件截图

6.2 任务 5

6.2.1 运行过程

```
18/07/20 12:11:50 INFO mapreduce.Job: Running job: job_1528693522781_6863
18/07/20 12:11:56 INFO mapreduce.Job: Job job_1528693522781_6863 running in uber mode : false
18/07/20 12:11:56 INFO mapreduce.Job:  map 0% reduce 0%
18/07/20 12:12:01 INFO mapreduce.Job:  map 13% reduce 0%
18/07/20 12:12:02 INFO mapreduce.Job:  map 40% reduce 0%
18/07/20 12:12:03 INFO mapreduce.Job:  map 57% reduce 0%
18/07/20 12:12:04 INFO mapreduce.Job:  map 77% reduce 0%
18/07/20 12:12:05 INFO mapreduce.Job:  map 92% reduce 0%
18/07/20 12:12:06 INFO mapreduce.Job:  map 100% reduce 0%
18/07/20 12:12:08 INFO mapreduce.Job:  map 100% reduce 100%
18/07/20 12:12:08 INFO mapreduce.Job: Job job_1528693522781_6863 completed successfully
```

图 10 任务 5（预测）运行过程截图

6.2.2 作业运行状态

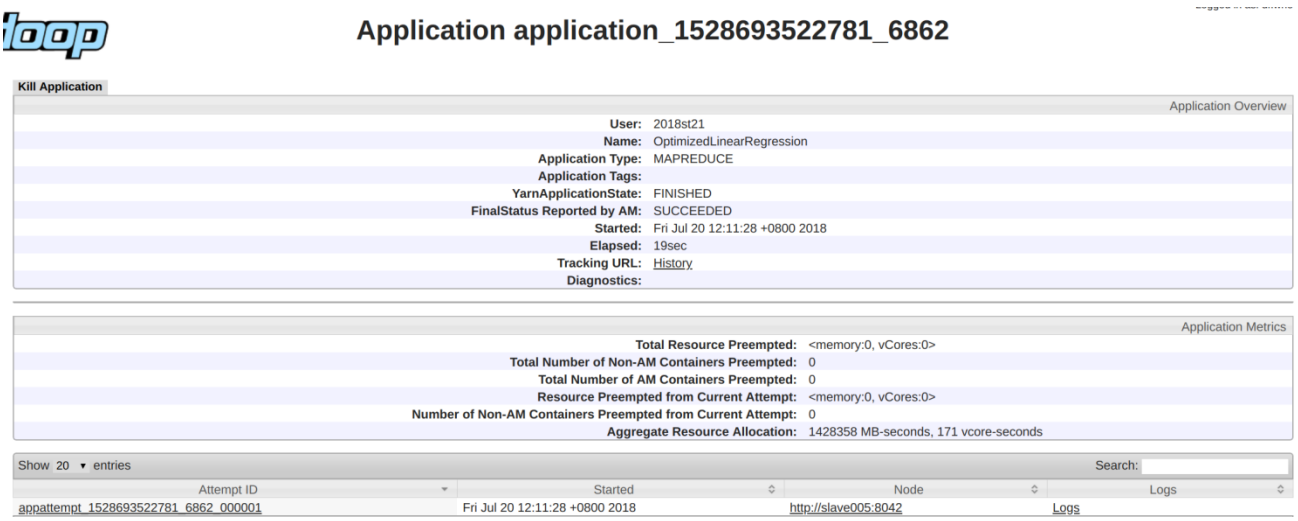

图 11 任务 5（预测）作业运行状态截图

6.2.3 查看输出文件

```
[2018st21@master01 OptimizedLinearRegression]$ cat RMSE/part-r-00000
rmse:   688.6901286220059
```

图 12 查看预测结果（RMSE 值）

```
00:00-01:00     7399.39
01:00-02:00     13329.95
02:00-03:00     6693.46
03:00-04:00     5500.15
04:00-05:00     5503.98
05:00-06:00     5424.74
06:00-07:00     6492.45
07:00-08:00     9592.15
08:00-09:00     22657.87
09:00-10:00     48502.89
10:00-11:00     61947.08
11:00-12:00     59005.14
12:00-13:00     39470.92
13:00-14:00     52578.41
14:00-15:00     58531.67
15:00-16:00     63521.08
16:00-17:00     61897.61
17:00-18:00     44683.26
18:00-19:00     23726.36
19:00-20:00     27408.15
20:00-21:00     35192.57
21:00-22:00     42871.27
22:00-23:00     39289.77
23:00-00:00     23534.19
```

图 13 查看预测结果输出文件

7 实验遇到的问题

7.1 在任务 4 的 Combiner 中仿照 k-means 算法的伪代码（如下图所示）合并均值时，默认接受到的 value 值为整型数 1，

```
reduce(ClusterID, [(p1,1), (p2,1), ...])
{
  pm = 0.0;
  n = 数据点列表[(p1,1), (p2,1), ...]中数据点的总个数;
  for i=0 to n
     pm += p[i];
  pm = pm / n;  // 求得这些数据点的平均值
  emit(ClusterID, (pm, n));
}
```

图 14 k-means 算法 Combiner 伪代码

但实际运行中会出现异常，即 value 值出现了小数，如下图所示：

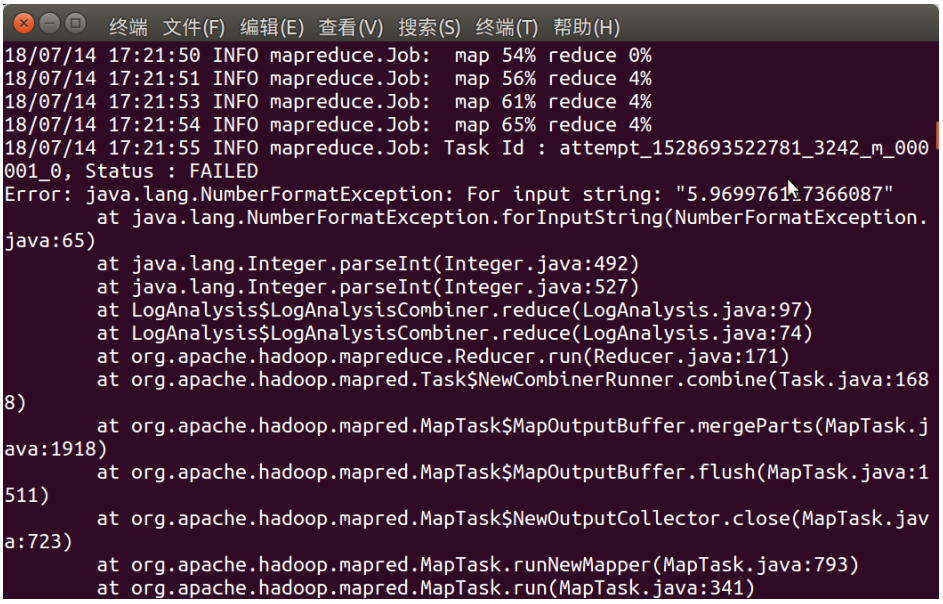

图 15 运行时出现小数异常

经查询资料发现原来 Combiner 可能会被反复调用，即一个 Combiner 合并的结果可能会再传给另一个 Combiner 继续合并，因此 Combiner 的输入输出格式必须一致，即 value 值都为浮点型数据。

7.2 任务 1 相同时间段状态码为 0 的不输出，例如：

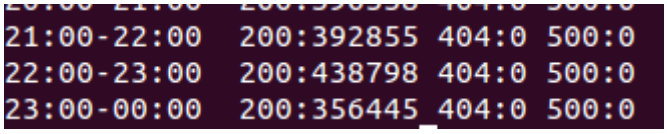

图 16 任务 1 输出结果

将不会输出 404:0 500:0。

原因：在 task1 的 reduce 环节，使用链表存储同一段时间的状态码及其出现频次，当状态码未曾出现时，map 阶段不会发送键值对，因此 reduce 节点就接受不到响应的状态码，也就不会输出。

解决方式：使用 count200,count404,count500 来保存当前时间段下各个状态码出现的次数。

7.3 计算 rmse 时一开始是在 combiner 部分将相同接口相同时间段的预测数据与真实数据作差，但结果错误。

原因：combiner 合并的是同一个 map 节点上的数据，若不在一个节点上不会合并，而预测数据

和真实数据不一定在一个 map 节点上执行 map 操作。

解决方式：改到 reduce 端执行预测数据与真实数据作差。

8 程序获取及运行方式

8.1 Github 访问地址

https://github.com/rubychen0611/LogAnalysis

8.2 任务 1-4 运行命令

hadoop jar LogAnalysis.jar LogAnalysis /data/task1/JN1_LOG/2015-09-08.log /user/2018st21/output1 /user/2018st21/output2 /user/2018st21/output3 /user/2018st21/output4

8.3 任务 5 运行命令

hadoop jar LogAnalysis.jar LogPredict /user/2018st21/trainingset /user/2018st21/result/OptimizedLinearRegression

9 实验总结

通过本次课程设计，我们实现了基础的海量日志数据的统计分析过程，以及通过对历史日志数据的分析建立预测模型，并基于 MapReduce 编程框架对接口访问量进行预测，进一步熟悉和掌握了 MapReduce 编程技能，对 MapReduce 技术在工业界的应用有了更深的了解。

[参考资料]

[1] Hadoop 的 MultipleOutputs 进行多文件输出. https://blog.csdn.net/u011007180/article/details/52260210
[2] 最小二乘法多项式拟合的 Java 实现. https://blog.csdn.net/funnyrand/article/details/46742561
[3] [Hadoop]输入路径过滤，通配符与 PathFilter. https://blog.csdn.net/sunnyyoona/article/details/53786397
[4] 多元线性回归与梯度下降. https://www.cnblogs.com/wzm-xu/p/4062266.html
[5] 模拟退火法.https://www.cnblogs.com/kaituorensheng/p/3583014.html
[6] 倪震、李千目、郭雅娟.面向电力大数据日志分析平台的异常检测算法集成算法.南京理工大学学报,2017,10.
[7] 常见的预测算法. https://blog.csdn.net/konglongaa/article/details/51453429